jamk.fi

SDN-verkot monitorointi ja ohjelmallinen ohjaus

Asmo Korkiatupa

Opinnäytetyö
Toukokuu 2016
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Korkiatupa, Asmo	Opinnäytetyö, AMK	18.05.2016
	Sivumäärä 47	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
SDN-verkot monitorointi ja ohjelmallinen ohjaus

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Hannu Luostarinen
Raija Hämäläinen

Toimeksiantaja(t)
JAMK Cyber Trust
Janne Alatalo

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Cyber Trust–hanke Jyväskylän ammattikorkea-koulussa. Työn tavoitteena oli tutkia SDN-verkossa liikkuvaa tietoliikennettä. Sen perusteella tuli ohjata verkon kuormaa sen eri osiin. Tavoitteena oli todistaa, että kuormanjakaminen SDN-verkoissa on mahdollista. Lisätavoitteena oli tutkia palvelunestohyökkäyksiä ja niiden estämiseen käytettäviä menetelmiä.

Työ toteutettiin VMware'n SDN-testbed-virtuaaliympäristössä. Verkkotopologia virtualisoitiin Mininet-ohjelmalla. Verkkoa monitoroitiin sFlow-RT ohjelmistolla ja verkon ohjaamiseen käytettiin Java-pohjaista Floodlight SDN-kontrolleria.

Opinnäytetyössä suunniteltiin ja toteutettiin kuormanjako-ohjelma, joka reagoi verkkoon tulevaan pakettiliikenteeseen. Ohjelma kirjoitettiin Python-ohjelmointikielellä. Ohjelmassa käytettiin sFlow-RT:n ja Floodlight-kontrollerin REST-rajapintaa. Ohjelmassa haettiin tietoa HTTP-GET-metodilla sekä lisättiin ja poistettiin uusia sääntöjä kytkimien Flow-tauluista HTTP–POST ja HTTP–DELETE-metodeilla.

Kirjoitettu ohjelma todisti, että kuormaa voidaan jakaa SDN-verkoissa halutulla tavalla. Palvelunestohyökkäyksien estämismenetelmiin voidaan lukea ohjelmistosuunnittelu, kaista ja kuormanjako, NFV-palvelut ja niihin reagointi SDN-verkoissa.

Avainsanat (asiasanat)
SDN, NFV, Floodlight, OpenFlow, sFlow, REST, JSON, Python

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Korkiatupa, Asmo	Bachelor's thesis	18.05.2016
		Language of publication: Finnish
	Number of pages: 47	Permission for web publication: x

Title of publication
SDN-networks monitoring and software defined routing

Degree programme
Software Engineering

Supervisor(s)
Luostarinen, Hannu
Hämäläinen, Raija

Assigned by
JAMK Cyber Trust
Alatalo, Janne

Abstract

This bachelor's thesis was assigned by Cyber Trust project in JAMK University of Applied Sciences. The goal of this thesis was to study traffic in SDN network and re-route network traffic to different parts of the network. The objective was to prove that traffic routing is possible in SDN networks. An additional goal of this thesis was to study Denial of Service attacks and the methods to prevent them.

The practical part of the Bachelor's thesis was implemented in VMware's virtual environment. The network topology was virtualized with Mininet emulator. The network was monitored with sFlow-RT program and Floodlight SDN controller was used to control the network.

A balance loader script was designed and programmed with Python programming language. The REST APIs of Floodlight controller and sFlow-RT were used in the program. The program fetched network traffic data from sFlow-RT's REST API with HTTP GET method and controlled the network routing through Floodlight's REST API with HTTP POST and DELETE methods.

The written script proved that network traffic can be re-routed in SDN networks as requested. The following methods can be noted in preventing Denial of Service attacks: Software design, NFV-services and applications, and reacting to DDoS attack in SDN network.

Keywords/tags (subjects)
SDN, NFV, Floodlight, OpenFlow, sFlow, REST, JSON, Python

Miscellaneous

Sisältö

Kuviot

Lyhenteet

API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
DNS	Domain Name System
DoS	Denial of Service
DDoS	Distributed Denial of Service
HTTP	Hypertext Transfer Protocol
IDP	Intrusion Detection and Prevention
IDPS	Intrusion Detection and Prevention System
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
JSON	Javascript Object Notation
MVC	Model View Controller
NFV	Network Function Virtualization
NV	Network Virtualization
REST	Representational State Transfer
SDN	Software-Defined Networking
TCP	Transmission Control Protocol
WAF	Web Application Firewall
XML	Extensible Markup Language

1 Työn lähtökohdat

1.1 Tehtävät ja tavoitteet

Opinnäytetyön tavoitteena oli tutkia SDN-verkon joustavuutta pakettien reitittämisessä ja tutkia uusia mahdollisuuksia palvelunestohyökkäyksien rajoittamiseksi. Palvelunestohyökkäykset ovat nykyään arkipäiväinen riesa yritysten, valtioiden ja järjestöjen elämässä. Opinnäytetyössä pyrittiin tutkimaan monitorointidataa sekä reagoimaan kuormaan tai ylimääräiseen liikenteeseen ohjelmallisin menetelmin.

Tämä työ oli jatkoa Pauli Raitasen (2015) opinnäytetyölle, Tietoliikennevoiden monitorointi Software-Defined Networkingissä. Hän tutki työssään kahta eri monitorointitekniikkaa SDN-ympäristössä.

Raitasen työssä keskityttiin seuraamaan verkon toimintaa sFlow- ja Netflow-tekniikoita käyttäen. Lisäksi siinä haluttiin selvittää, millä tavalla SDN-kontrollerin ja sFlow-monitorointipalvelimen on mahdollista keskustella keskenään, jotta OpenFlow-pohjainen SDN-kontrolleri pystyisi reagoimaan verkossa tapahtuviin muutoksiin.

Tässä opinnäytetyössä pyrittiin seuraamaan verkon toimintaa sFlow-monitorointipalvelimen avulla ja vaikuttamaan OpenFlow-kontrollerin kautta verkon toimintaan saadun datan perusteella.

Opinnäytetyön tavoitteena oli vaikuttaa verkon toimintaan analytiikkadatan perusteella. Datan haku ja verkon ohjaus suoritettiin sFlow-monitorointipalvelimen ja OpenFlow-kontrollerin REST-rajapintaa käyttäen. Tavoitteena oli todistaa, kuinka verkossa pystytään jakamaan kuormaa verkon eri osiin käytännössä.

Lisätavoitteena oli tutkia palvelunestohyökkäyksiä ja tapoja niiden estämiseksi ja esittää tästäkin käytännön esimerkki, jos aikaa sille vielä jää.

1.2 Toimeksiantaja

Toimeksiantajana toimi Cyber Trust -hanke Jyväskylän ammattikorkeakoulussa. Cyber Trust on DIGILE Oy:n rahoittama Suomen laajuinen hanke. Siinä on mukana useita tunnettuja yrityksiä kuten Elisa, Erikson, F-Secure, Fujitsu Finland, Intel Finland, Digile

jne. Jyväskylän ammattikorkeakoulun Cyber Trust on vain yksi tutkijaorganisaatio osana suurempaa kokonaisuutta. Muita ovat esimerkiksi Aalto-yliopisto, Helsingin yliopisto, Jyväskylän yliopisto, Oulun yliopisto. (Cyber Trust n.d.)

Vuosien varrella olemme tulleet yhä riippuvaisemmiksi tietoverkoista. Tietoverkkojen kehittyessä niihin kohdistuu yhä enemmän ennalta arvaamattomia riskejä. Näihin riskeihin olisi syytä varautua ja tutkia tulevaisuuden tekniikoita jo etukäteen. (Cyber Trust n.d.)

Digilen tavoitteena on luoda Suomesta edelläkävijä tietoverkko-osaamisen ja kyberturvallisuuden osa-alueella. Cyber Trust -hanke käynnistettiin toukokuussa 2015. Sen tavoitteena on tutkia uusia verkkoteknologioita, niiden turvallisuutta ja uusia ohjelmallisia ratkaisuja. (Cyber Trust n.d.)

2 Software defined network - SDN

2.1 Yleistä

Perinteiset tietoverkkoarkkitehtuurit eivät vastaa nykypäivän yritysten, toimittajien ja loppukäyttäjien vaatimuksia. Kiitos laajan teollisuuden ponnistelun, jota on johtanut Open Networking Foundation, SDN on muuttuva tietoverkkoarkkitehtuuri. (Software-Defined Networking: The New Norm for Net-works 2012, 2.)

SDN arkkitehtuurissa hallinta- ja datapinnat on erotettu toisistaan, tietoverkon äly ja logiikka ovat loogisesti keskitettyjä ja perustana oleva verkkoarkkitehtuuri on yksinkertaistettu kaikille sovelluksille. Tuloksena yritykset ja toimittajat saavat ennennäkemättömän mahdollisuuden tietoverkon ohjelmointiin, automatisointiin ja määritykseen. Tämä mahdollistaa erittäin skaalautuvien ja joustavien verkkojen rakennuksen, jotka sopeutuvat helposti yritysten muuttuviin tarpeisiin. (Software-Defined Networking: The New Norm for Net-works 2012, 2.)

ONF on voittoa tavoittelematon yhtymä, joka johtaa SDN-verkkojen ja siihen liittyvien kriittisten standardien kuten OpenFlow'n kehitystä. OpenFlow tarjoaa tarvittavan rakenteen kontrollitason ja välitystason kommunikaatioon. Se on ensimmäinen standardisoitu rajapinta, joka on erityisesti suunniteltu SDN:n tarpeisiin. (Software-Defined Networking: The New Norm for Net-works 2012, 2.)

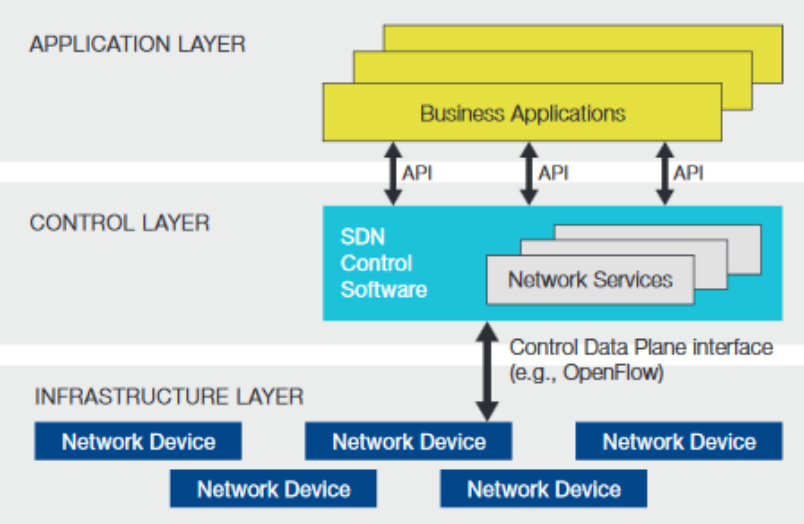
OpenFlow-perusteista SDN arkkitehtuuria tuotetaan juuri nyt useille verkkolaitteille ja ohjelmistoille, mikä tarjoaa huomattavaa hyötyä yrityksille ja jakelijoille. Näitä hyötyjä ovat mm. keskitetty hallinta, parannettu automatisointi, nopea innovatiivinen kyky tuottaa verkko-ominaisuuksia ja palveluita ilman tarvetta määrittää asetuksia yksittäisille laitteille tai odottaa toimittajan julkaisuja. Näillä ratkaisuilla saadaan parannettua myös verkon luotettavuutta ja turvallisuutta, koska verkonhallinta on keskitetty ja automatisoitu. (Software-Defined Networking: The New Norm for Net-works 2012, 2.)

2.2 Tekniikka

Ohjelmallisesti määritetty verkko SDN on kasvava verkkoarkkitehtuuri. Siinä kontrollitaso erotetaan välitystasosta, jolloin saadaan kontrollitasosta suoraan ohjelmoitava. Kontrollitaso oli aikaisemmin tiukasti sidottuna yksittäiseen verkkolaitteeseen. Nyt

kun kontrollitaso voidaan eriyttää ohjelmoitaviin laitteisiin, voidaan verkon infrastruktuuria käyttää sovelluksissa ja verkkopalveluissa. Sovellukset tai palvelut voivat kohdella verkkoa joko loogisena tai virtuaalisena kokonaisuutena. (Software-Defined Networking: The New Norm for Net-works 2012, 7.)

Kuviossa 1 nähdään looginen näkymä SDN-arkkitehtuurista. Verkon älykkyys on loogisesti keskitetty ohjelmallisiin SDN-kontrollereihin, kontrollerit pitävät globaalin näkymän verkosta. Tuloksena verkko esiintyy sovelluksille yhtenä loogisena kytkimenä. SDN:n avulla yritykset saavat tarjoajariippumattoman kontrollin koko tietoverkkoon yhdestä loogisesta sijainnista, joka hienosti yksinkertaistaa tietoverkon tyylin ja toiminnan. (Software-Defined Networking: The New Norm for Net-works 2012, 7.)



Kuvio 1. SDN-arkkitehtuuri (Software-Defined Networking: The New Norm for Networks 2012, 7).

Ehkä tärkeimpänä ominaisuutena verkko operaattorit ja pääkäyttäjät voivat ohjelmallisesti määrittää tämän yksinkertaistetun verkon toiminnan helpommin, verrattaessa siihen, että pääkäyttäjän pitäisi käsin kirjoittaa kymmeniä tuhansia rivejä määrityksiä tuhansille laitteille, jotka ovat sijoiteltu ympäri tietoverkkoa. Tämän lisäksi ohjelmallisesti määritetyn verkon toimintaa voidaan muuttaa reaaliajassa nojautuen SDN-kontrollerin keskitettyyn älykkyyteen. Tästä johtuen verkkoon pystytään toimittamaan uusia ohjelmistoja tai tietoverkkopalveluita todella lyhyellä aikavälillä, joko tunneissa tai päivissä. Nykyään siihen saattaa kulua viikkoja tai jopa kuukausia. Kun

keskitetään verkon kontrollitaso, SDN tarjoaa tietoverkon hallitsijoille joustavuutta määrittää, hallita, turvata ja optimoida verkon resursseja automatisoiduilla SDN-ohjelmistoilla. Sen lisäksi verkon hallitsijat voivat kirjoittaa nämä tarvittavat ohjelmistot itse, jolloin heidän ei tarvitse odottaa palveluntarjoajien tekemiä ohjelmistoja. (Software-Defined Networking: The New Norm for Net-works 2012, 8.)

Sen lisäksi, että verkko pystytään yksinkertaistamaan helpoksi hallittavaksi kokonaisuudeksi, SDN tarjoaa useita eri rajapintoja, joilla verkon muokkaaminen tarpeiden mukaan on helppoa. Esimerkiksi SDN-arkkitehtuuri tekee helpoksi määrittää ja vahvistaa jatkuvat politiikat sekä langalliseen että langattomaan verkkoon vaikkapa oppilaitoksella. (Software-Defined Networking: The New Norm for Net-works 2012, 8.)

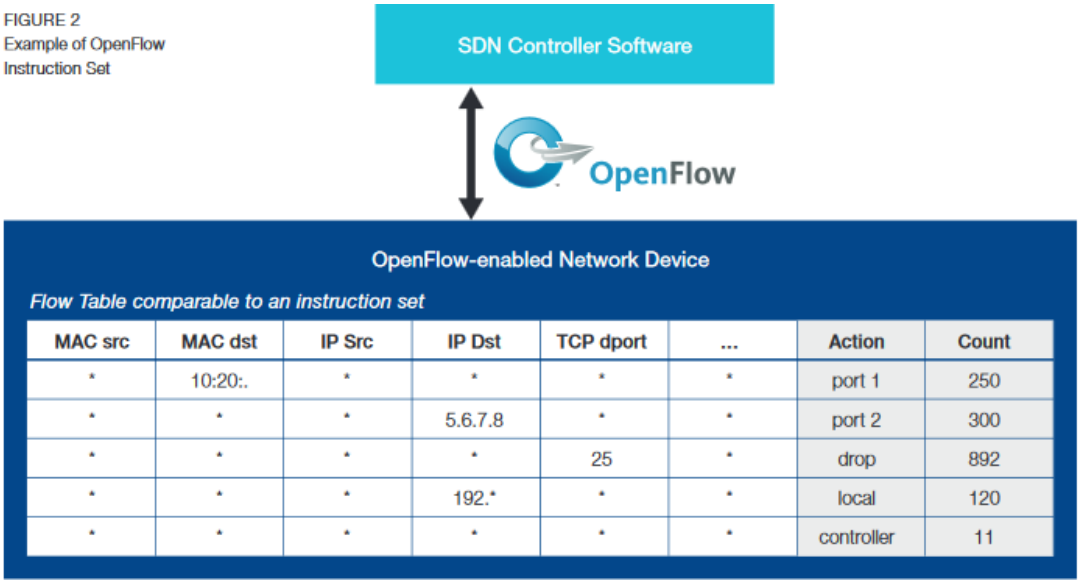
Tässä opinnäytetyössä infrastruktuuritaso virtualisoitiin Mininet-ohjelmiston avulla, kontrollitasossa käytettiin Floodlight SDN-kontrolleria ja sovellustasossa esiteltiin kuormanjako-ohjelmisto, jolla voitiin jakaa kuormaa verkon eri osiin.

2.3 OpenFlow

OpenFlow on ensimmäinen standardisoitu kommunikaatiorajapinta, joka on määritetty kontrollitason ja välitystason väliin SDN-arkkitehtuurissa. OpenFlow tarjoaa suoran pääsyn tietoverkon laitteiden fyysisiin ja virtuaalisiin välityspintoihin, kuten kytkimiin ja reitittimiin, sekä mahdollistaa niiden muokkaamisen. Avoimen rajapinnan puuttuminen on syy, joka on johtanut nykypäivän tietoverkkojen yhtenäiseen, suljettuun ja keskustietokonemaiseen luonteeseen. Mikään muu standardi protokolla ei tee, mitä OpenFlow tekee, ja OpenFlow'n tyylinen protokolla tarvitaan, jotta tietoverkon ohjaus saadaan pois verkon kytkimiltä loogisesti keskitettyyn hallintaohjelmistoon. (Software-Defined Networking: The New Norm for Net-works 2012, 8.)

OpenFlow-protokollaa voidaan verrata prosessorin käskykantaan. Kuten kuviossa 2 näytetään, protokolla määrittää peruskäskykannan, jota voidaan käyttää ulkoisessa ohjelmasovelluksessa, kun ohjelmoidaan verkkolaitteiden välityspintaa. Samantyylistä käskykantaa käytetään prosessorin käskyjen ohjelmoimiseen tietokonejärjestelmässä. (Software-Defined Networking: The New Norm for Net-works 2012, 8.)

FIGURE 2
Example of OpenFlow
Instruction Set



MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Kuvio 2. OpenFlow-käskykanta (Software-Defined Networking: The New Norm for Net-works 2012, 8).

Tätä käskykantaa voidaan käyttää jokaiselle verkon laitteelle. Verkon reitittämisessä tähän laitteen käskykantaan luodaan lauseita ja ohjataan verkon toimintaa halutulla tavalla. Vaikkapa luodaan sääntö, että kaikki porttiin x tulevat paketit ohjataan portti y:n kautta. Liikennettä voidaan ohjata myös MAC- tai IP-osoitteella.

2.4 SDN-kontrollerit

Luvussa 2.4.1 esitetyt kontrollerit ovat avoimen lähdekoodin ohjelmistoja. Kontrollerit esitellään jokseenkin aikajärjestyksessä. SDN-kontrollereita on olemassa paljon. Tässä luvussa esitetään vain muutamia, koska niiden kaikkien läpikäyminen olisi opinnäytetyön kannalta turhaa. Sanottakoon kuitenkin, että kaupallisia SDN-kontrollereita tarjoavat mm. Cisco, HP ja VMware.

Avoimet vaatimukset ovat avainkomponentti SDN-arkkitehtuurissa, avoimen lähdekoodin SDN-kontrollerit ovat kehittyneet vuosien mittaan. (Wilkins 2014).

2.4.1 NOX

On olemassa useita avoimen lähdekoodin SDN-kontrollereita, jotka ovat olleet saatavilla jonkin aikaa. Ensimmäinen suuren suosion saavuttanut OpenFlow-kontrolleri oli

NOX. Se oli ensimmäinen laatuaan ja alkukipinä SDN-kontrollereiden kehityksessä, mutta ei kuitenkaan tullut kovinkaan käytetyksi. NOX:n kompastuskiveksi koitui huono dokumentaatio ja ohjelmointiin käytettävä C++ kieli. (Wilkins 2014.)

2.4.2 POX

NOX:n seuraaja POX oli rakennettu käyttäjäystävällisemmäksi. Verrattuna NOXiin POXilla on helpompi kehitysympäristö sekä hyvin kirjoitettu ja dokumentoitu API. Se tarjoaa myös web-pohjaisen käyttöliittymän. POX on kirjoitettu Python-ohjelmointikielellä. (Wilkins 2014.)

2.4.3 Beacon

Seuraava suuri askel avoimen lähdekoodin kontrollereissa tuli Beacon-kontrollerin mukana. Beacon on erittäin hyvin kirjoitettu ja organisoitu SDN-kontrolleri. Beacon on kirjoitettu Javalla ja integroitu suuresti Eclipse-sovelluskehittimeen. Beacon oli ensimmäinen kontrolleri, joka teki mahdolliseksi tavallisen ohjelmoijan luoda toimiva SDN-ympäristö. Ympäristö oli kuitenkin rajoitettu tähtitopologioihin. (Wilkins 2014.)

2.4.4 Floodlight

Beaconin jälkeen tuli Floodlight. Käytännössä tämä oli Beaconista otettu kopio, jota hallinnoi Big Switch Networks. Vaikkakin Floodlight-kontrollerin alku perustuukin Beaconiin, se rakennettiin käyttäen Apache Ant -työkalua, joka tekee Floodlightin kehittämisestä helpompaa ja joustavampaa. Floodlightilla on erittäin aktiivinen kehittäjäryhmä ja suuri määrä ominaisuuksia, joista on helppo räätälöidä kullekin organisaatiolle toimiva ratkaisu. Käyttöliittymä on saatavilla web-pohjaisena sekä Javapohjaisena, ja useimpia sen ominaisuuksia voidaan hallinnoida REST API:n välityksellä. (Wilkins 2014.)

2.4.5 Ryu

Ryu on SDN-kontolleri, joka on kirjoitettu Python-ohjelmointikielellä. Ryu on erittäin muokattavissa oleva kevyt ohjelmistoratkaisu. Ryu:ssa kontrolleriskriptejä on useita ja logiikka voidaan kirjoittaa kaikki itse. Tämä on mielenkiintoinen ratkaisu ja joissain

määrin erittäin hyvä idea. Kontrollerista saadaan erittäin kevyt, koska siellä ei ole mitään ylimääräistä.

Kuitenkin joissain tapauksissa Ryu on huono ratkaisu, se havaittiin tätä opinnäytetyötä tehtäessä. Vaikkakin Ryu:n arkkitehtuuri mahdollistaa suuren muokattavuuden, valmiita kontrolleriskriptejä on suhteellisen vähän. Valmiit kontrolleriskriptit eivät tue monimutkaisia topologita, kuten tässä opinnäytetyössä käytettävää ”tikapuu”-mallia. Jos työssä olisi käytetty Ryu:n REST router kontrolleria, olisi kontrollerin logiikka jouduttu kirjoittamaan käsin.

Opinnäytetyössä haluttiin kuitenkin dynaaminen ratkaisu — ratkaisu, joka käyttäisi hyväkseen REST-rajapintoja. Tästä syystä työssä päädyttiin käyttämään Floodlight-kontrolleria. Ja mihin Ryu:n konfiguroinnissa oli mennyt kolme tuloksetonta päivää, Floodlight-kontrollerilla toimivan verkon konfigurointi onnistui automaattisesti.

3 sFlow

3.1 Yleistä

Yritysten tärkeät sovellukset ovat tulleet hyvin riippuvaiseksi verkosta, pieninkin muutos verkon käytössä voi vaikuttaa verkon suorituskykyyn ja luotettavuuteen. Sillä saattaa olla suora vaikutus yrityksen toimintaan, tietoverkkojen ylläpitopalveluiden kustannuksella. (Traffic Monitoring using sFlow n.d., 1.)

sFlow tarjoaa ennennäkemättömän näkyvyyden verkon käyttöön ja aktiivisiin reitteihin, jopa tämän päivän nopeisiin ja monimutkaisiin verkkoihin. Se tarjoaa tarvittavan datan verkkojen tehokkaaseen kontrolliin ja verkon käytön hallinnointiin. Tämä varmistaa, että tietoverkkopalvelut tarjoavat kilpailullisen edun. (Traffic Monitoring using sFlow n.d., 1.)

Esimerkkejä sovelluksista, jotka käyttävät sFlow-dataa:

- Tietoverkko-ongelmien tunnistaminen, diagnosointi ja korjaaminen
- Ajantasainen ruuhkan hallinta
- Tietoverkon käyttö laskutukseen ja takaisin maksuun
- Verkkopolkujen tarkastus luvattoman verkon käytön havaitsemiseksi ja palvelunesto-hyökkäyksien lähteiden jäljitys
- Reititysprofilointi ja reititysoptimointi
- Kapasiteettisuunnittelu

(Traffic Monitoring using sFlow n.d., 1.)

sFlow on näytteenottoteknologia, joka vastaa avainvaatimuksiin verkon liikenteen monitorointiratkaisuna:

- **sFlow tarjoaa koko verkon kattavan näkymän** tietoverkon käytöstä ja aktiivisista reiteistä. Se on skaalautuva tekniikka verkon liikenteen mittaamiseen, keräämiseen, tallentamiseen ja analysointiin. Tämä mahdollistaa kymmenien tuhansien rajapintojen monitoroinnin samasta paikasta.
- **sFlow on skaalautuva**, se mahdollistaa linkkien monitoroinnin 10 Gb/s nopeuksilla ja sen yli. Ilman, että se vaikuttaa internetreittien tai kytkimien suorituskykyyn. Se ei myöskään lisää merkittävää verkkokuormaa.

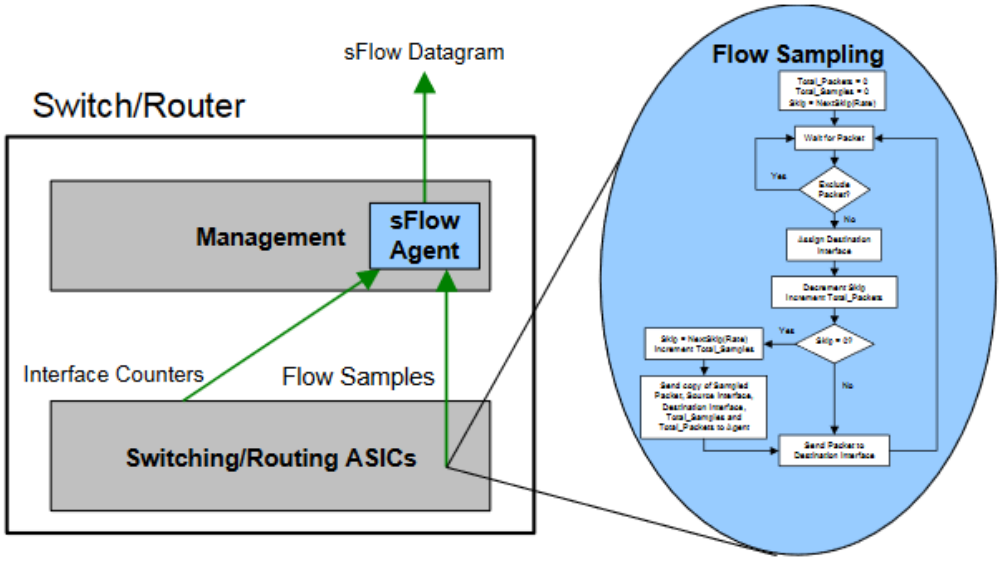
- **sFlow on halpa ratkaisu**. Se on implementoitu laajaan valikoimaan laitteita, yksinkertaisista kytkimistä monimutkaisiin huippuluokan kytkimiin.
- **sFlow on toimiala standardi** ja kasvava määrä yrityksiä tarjoaa laitteita sFlow-tuella.

(Traffic Monitoring using sFlow n.d., 1.)

3.2 Mikä on sFlow?

sFlow on näytteenottoteknologia, joka on sulautettu useiden valmistajien reitittimiin ja kytkimiin. Se tarjoaa mahdollisuuden tarkkailla jatkuvasti sovellustason liikennevirtoja yhtäaikaisesti kaikissa rajapinnoissa. (Traffic Monitoring using sFlow n.d, 2)

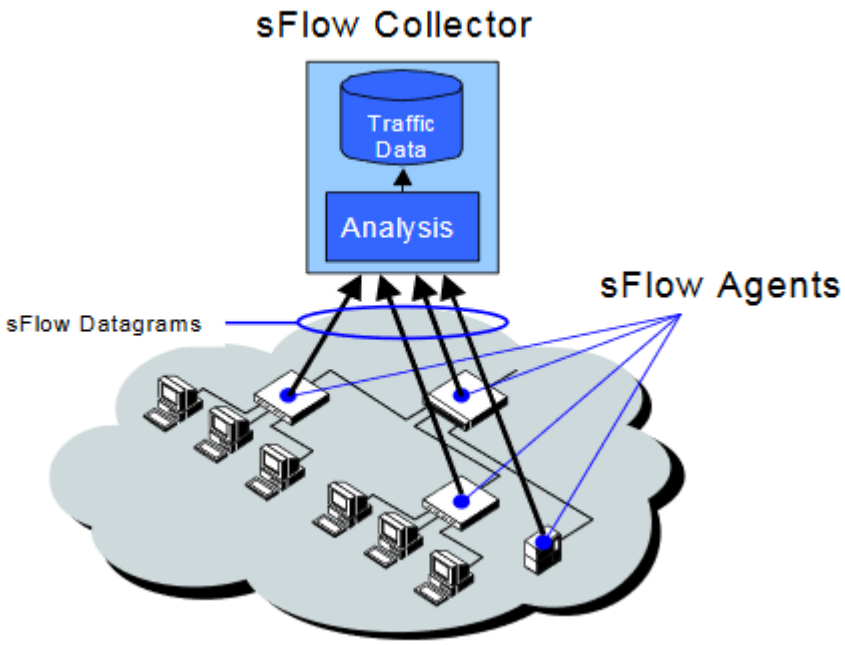
sFlow Agentti on ohjelmistoprosessi, joka pyörii osana tietoverkon hallintaohjelmistoa laitteen sisällä. Tämä on nähtävissä kuviossa 3. Se yhdistää rajapinnan laskurin ja virtausnäytteet ja lähettää ne sFlow-kerääjälle. Pakettien näytteenotto suoritetaan yleensä kytkimien tai reitittimien ASIC:n toimesta, mikä tarjoaa nopean suorituskyvyn. Näytteenotossa saatujen pakettien reititystaulujen merkinnät otetaan myös talteen. (Traffic Monitoring using sFlow n.d., 2.)



Kuvio 3. sFlow Agentti sulautettuna reitittimeen (Traffic Monitoring using sFlow n.d., 2).

sFlow Agentti itsessään tekee hyvin vähän prosessointia. Se yksinkertaisesti paketoi datan ja lähettää sen välittömästi sFlow-kerääjälle. Välitön välitys minimisoi sFlow Agentin tarvitseman muistin ja prosessoritehon määrän. (Traffic Monitoring using sFlow n.d., 2.)

Kuviossa 4 nähdään sFlow-järjestelmän peruselementit. sFlow Agentit, jotka on levitetty ympäri verkkoa, lähettävät jatkuvana tietovirtana sFlow-paketteja sFlow-kerääjälle. sFlow-kerääjässä paketit analysoidaan ja ne tuottavat rikasta, reaaliaikaista, koko tietoverkon kattavaa näkymää tietoliikennevirroista. (Traffic Monitoring using sFlow n.d., 3.)



Kuvio 4. sFlow Agentit ja kerääjä (Traffic Monitoring using sFlow n.d., 3).

sFlow monitoroinnilla nopeissa, reititetyissä tietoverkoissa on seuraavat avainominaisuudet:

- **Tarkka.** Koska näytteenotto on niin yksinkertaista, se voidaan tehdä rautatasolla, joka toimii nopeasti. Lisäksi sFlow-järjestelmä on suunniteltu niin, että mittauksen tarkkuus voidaan määrittää itse.

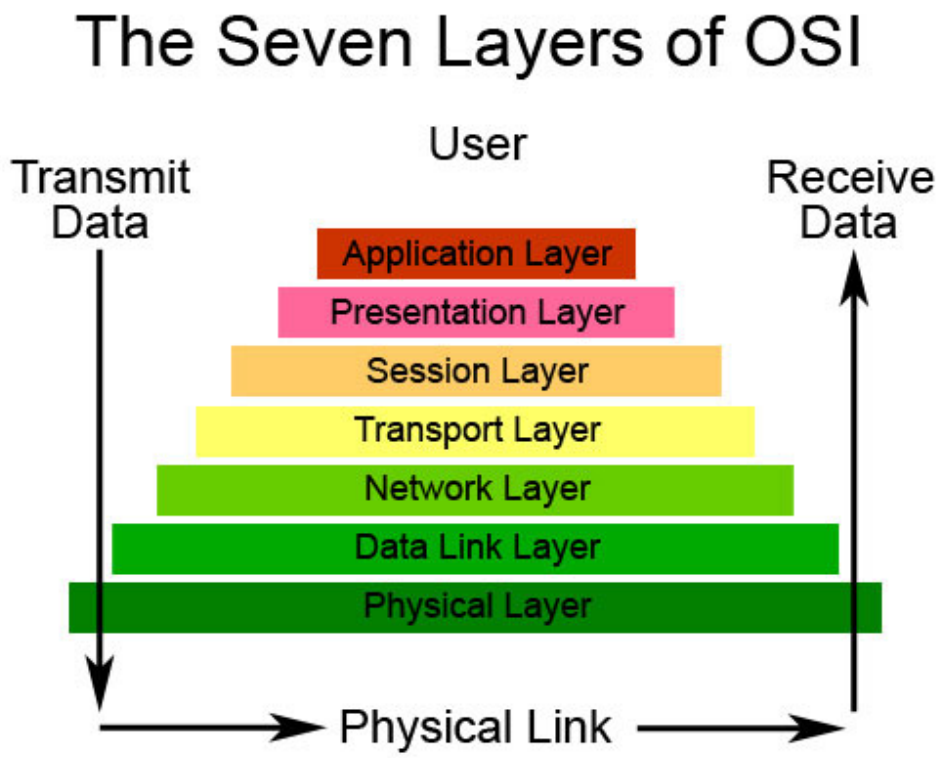
- **Yksityiskohtainen.** Kokonaiset pakettien otsikoinnit ja reititystiedot mahdollistavat yksityiskohtaisen analyysin kuvion 5 OSI-mallin tasojen 2-7 tietoliikennevirtoihin.

- **Skaalautuva.** Toiminnan yksinkertaisuus mahdollistaa suurien nopeuksien tietovirtojen tutkimisen, ja tuhansia laitteita voidaan monitoroida yksittäisellä sFlow-kerääjällä.

- **Halpa.** sFlow Agentti on yksinkertainen implementoida ja ei juurikaan lisää kustannuksia reitittimeen tai kytkimeen.

- **Ajankohtainen.** sFlow-kerääjällä on aina ajankohtainen näkymä verkon liikenteestä. Ajankohtainen tieto on erityisen tärkeää, jos verkkoa tulee hallinnoida reaaliaikaisesti, esimerkiksi palvelunestohyökkäyksen puolustamiseen.

(Traffic Monitoring using sFlow n.d., 3.)



The seven layers of the Open System Interconnection (OSI) Model.

Kuvio 5 OSI-malli

4 Network Functions Virtualisation – NFV

4.1 Yleistä

SDN-verkkojen yhteyteen tarjotaan usein NFV-palveluita. NFV virtualisoi OSI-mallin tasojen 2-7 palveluita kuten palomuuri tai IDPS. OSI-malli on nähtävissä kuviossa 5.

Jos verkon järjestelmänvalvojat voivat luoda virtuualiverkkoja osoittamalla ja klikkaamalla, miksi he eivät voisi kytkeä palomuuria tai IDS/IPS-palvelua päälle samalla tavalla? Tämä on se mitä NFV-palvelut mahdollistavat. Jos on tietty tunneli, joka on reititetty verkkoinfrastruktuurin läpi, voi palomuurin tai IDS/IPS palvelun lisätä juuri siihen tunneliin. (Garrison 2014.)

NFV toimii suuritehoisilla x86-alustoilla, ja se mahdollistaa käyttäjien asettaa palveluita valittuihin tunneleihin tietoverkossa. Tavoite on luoda ihmisille mahdollisuus tehdä palveluprofiili tietoverkkoon ja käyttää x86-alustan laskentatehoa rakentamaan suodatus tietoverkon päälle, joka koostuu erinäisistä virtuaalipalveluista. (Garrison 2014.)

NFV vähentää ylivarustelun tarvetta. Ennemmin kuin ostaisi ison palomuurin tai IDS/IPS-laatikot, jotka voisivat käsitellä koko tietoverkon. Asiakas voi ostaa palvelut vain niihin paikkoihin, jotka niitä tarvitsevat.(Garrison 2014.)

4.2 Turvallisuuden rooli SDN-verkoissa ja NFV-palveluissa

Vaikkakin SDN ja NFV ovat kuumia aiheita puhuttaessa tietoverkoista ja ne pakottavat kauppiaat, jotka myyvät verkkotuotteita tai sovelluksia, uudelleen suunnittelemaan joitakin heidän palveluitaan. Tietoturvatoimittajat ovat työskennelleet virtuaalisoitujen ympäristöjen sopeutuvien uhkien tunnistus- ja lievennysratkaisujen parissa jo vuosia, yleensä rakentaen virtuaalisia ilmentymiä perinteisistä rauta- tai ohjelmistotuotteista. (Wilson 2015, 3.)

Syy tähän on hyvin yksinkertainen. Kun palvelimien virtualisointi esiteltiin, kaikki niitä tarjoavat toimittajat tajusivat, että tarvittavaan tietoturvatasoon oli hankala päästä. He tarvitsivat tavan turvatakseen liikennettä, joka kulkee virtuaalikoneiden välillä. Tämä liikenne oli nyt näkymätöntä heidän olemassa oleville suodatuspalveluilleen.

Ostajat kääntyivät tietoturvayhtiöiden puoleen ja halusivat samanlaiset verkonhallinta työkalut kuin oli jo rautapuolella olemassa, kuten palomuuri, virustentorjunta, tunkeutumisen esto jne. Virtuaalisen turvallisuuden soveltaminen oli syntynyt. (Wilson 2015, 3.)

Vaikkakin SDN-kontrolleri on SDN-verkon aivot, oikea teho löytyykin sen toiminnan liittämisestä suureen valinnanvaraan NFV-palveluita. Nämä yhdessä mahdollistavat kustomoitavien palveluiden rakentamisen erittäin helpolla tavalla. Se on SDN-verkkojen ja NFV-palveluiden ketterä etu. Pelkästään turvallisuudessa on olemassa tusina ydintoimintoja, joista tulee omia NFV-palveluita. Tällaisia ovat esimerkiksi palomuuri, web-suodatus, virusskannaus, IDS, IPS, DDoS suodatus ja WAF. Nämä toiminnot voidaan liittää yhteen rakennettaessa yksittäistä räätälöityä palvelua asiakkaan tarpeisiin. (Wilson 2015, 4.)

5 Palvelunestohyökkäykset

5.1 Yleistä

Palvelunestohyökkäyksen tarkoitus on estää palvelun tai verkkosivuston normaali toiminta. Tämä suoritetaan lähettämällä palvelulle valtava määrä pyyntöjä, joihin palvelin yrittää vastata, kunnes päästään siihen pisteeseen, että palvelin ei pysty enää näitä kaikkia pyyntöjä käsittelemään. Tällöin palvelin vastaa normaalin käyttäjän pyyntöihin erittäin hitaasti tai ei ollenkaan. Palvelunestohyökkäys voi ottaa kohteekseen tietoverkon välityspinnan tai sovelluspinnan. Välityspinnalle voidaan lähettää massiivinen määrä TCP/IP-pyyntöjä tai vastaavasti sovelluspinnalle DNS- tai HTTP-pyyntöjä. (Denial of service attack n.d.)

5.2 Kuinka palvelunestohyökkäykset ovat muuttuneet?

Palvelunestohyökkäykset ovat yksi vanhimmista hyökkäysmetodeista, mutta ne ovat kehittyneet tärkeillä tavoilla (Denial of service attack n.d.).

Hyökkääjän on mahdollista tehdä ja toimittaa suuria massoja automaattisia hyökkäysrobotteja hajautettuna useisiin kymmeniin, satoihin tai jopa tuhansiin tietokoneisiin ympäri maailmaa. Usein tietokoneiden omistajat eivät edes tiedä, että heidän koneeseen on tartutettu hyökkäysrobotti, jota käytetään hajautettuun palvelunestohyökkäykseen. Tämä on johtanut DDoS hyökkäyksiin, jotka ovat paljon suurempia kuin oli mahdollista DoS-hyökkäyksille menneisyydessä. (Denial of service attack n.d.)

DDoS-hyökkäysten tekijät ovat kasvavissa määrin muuttaneet heidän keskittymistään ylemmäs internetprotokollahierarkiassa. Uusimmat hyökkäykset keskittyvät sovellustasolle, jossa DDoS-hyökkäys on vaikeampi havaita. (Denial of service attack n.d.)

Palvelunestohyökkäystä käytetään kasvavissa määrin hämäyksenä, jotta saadaan huomion keskittymään toisaalle. Kun huomio on kiinnitetty palvelunestohyökkäyksen estämiseen, muut yksittäiset hyökkäykset yrittävät etsiä hyväksikäytettäviä reikiä web-sovellusten datasuojausmenetelmistä mm. SQL-injektioiden avulla. (Denial of service attack n.d.)

5.3 Erilaiset palvelunestohyökkäykset

5.3.1 Palvelunestohyökkäys – DoS

DoS-hyökkäys on hyökkäys, jossa massiivinen määrä pyyntöjä lähetetään tiedetylle kohteelle verkossa. Kun kohteena oleva palvelu saavuttaa maksimaalisen suorituskykynsä pyyntöjen käsittelyyn, se ei pysty enää vastaamaan pyyntöihin, joita sille lähetetään. Joten jos DoS-hyökkäyksen suorittaa vaikkapa troijalainen virus, joka lähettää jatkuvasti massiivisen määrän pyyntöjä kohteeseen. Kohde pystyy käsittelemään vain nämä viruksen lähettämät pyynnöt, ja pyynnöt, jotka tulevat toisista osoitteista käsitellään viiveellä tai jäävät käsittelemättä kokonaan. (Podrezov 2004.)

DOS hyökkäykset ovat tulleet aika suosituksi nykypäivänä, koska ovat useissa tapauksissa suhteellisen tehokkaita. Esimerkiksi vuonna 2002 oli DOS-hyökkäys muutamaan IRC-palvelimeen ja usea käyttäjä ei pystynyt enää yhdistämään kyseisiin palvelimiin. Samana vuonna oli hyökkäyksiä muutamiin suuriin sivustoihin, mikä häiritsi normaalia sivuston käyttämistä ja aiheutti taloudellisia tappioita yrityksiin, jotka omistivat ne. (Podrezov 2004.)

Jotkin madot suorittavat niin sanottuja poliittisia DOS hyökkäyksiä. Esimerkiksi Yaha mato yritti DoS hyökätä Pakistanin hallituksen verkkosivuille. (Podrezov 2004.)

Tässä viitattu Podrezovin (2004) artikkeli on uhkakuvaus F-Securen verkkosivuilta vuodelta 2004. Uhkakuvauksesta voidaan huomata, että palvelunestohyökkäykset ovat olleet keskuudessamme jo kauan, eikä niihin vieläkään ole keksitty yksiselitteistä ratkaisua hyökkäysten estämiseksi.

DoS hyökkäyksessä tekijä käyttää yhtä internetyhteyttä joko käyttääkseen hyväksi ohjelmiston haavoittuvuutta tai tukkiakseen kohteen tekaistuilla pyynnöillä. Yleensä hyökkääjän motiivina on ylikuormittaa palvelimen resurssit. (Denial of service attacks n.d.)

5.3.2 Hajautettu palvelunestohyökkäys – DDoS

Hajautettu palvelunestohyökkäys eli DDoS laukaistaan useilta yhdistetyiltä laitteilta, jotka on sijoiteltu ympäri maailmaa. Nämä usean henkilön, usean laitteen hyökkäykset ovat yleensä hankalampia torjua johtuen useimmiten jo hyökkäykseen käytettävien laitteiden suuresta määrästä. Toisin kuin yksittäisestä lähteestä tulevissa DoS-hyökkäyksissä DDoS-hyökkäyksillä on tapana ottaa kohteekseen verkkoinfrastruktuuri yrityksenä kyllästää se massiivisella määrällä pyyntöjä. (Denial of service attacks n.d.)

DDoS-hyökkäykset eroavat myös niiden toteutustavassa. Yleisesti ottaen DoS hyökkäykset käynnistetään käyttäen kotitekoisia skriptejä tai DoS-työkaluja, kun taas DDoS-hyökkäykset käynnistetään ns. ”bottiverkoista” eli isoista klustereista yhdistettyjä laitteita. ”Bottiverkoilla” tarkoitetaan verkon laitteita, kuten tietokoneita, puhelimia tai reitittimiä, jotka on saastutettu haittaohjelmilla. Nämä haittaohjelmat mahdollistavat verkkolaitteen käyttämisen osana hajautettua palvelunestohyökkäystä. (Denial of service attacks n.d.)

5.4 Palvelunestohyökkäysten estämiseen käytettävät menetelmät

5.4.1 Kaista ja kuormanjako

Helpoin tapa, mutta myös kallis tapa palvelunestohyökkäykseen varautumiseen on kaista. Palvelunestohyökkäys on kapasiteetin peli. Jos on olemassa vaikkapa 10 000 järjestelmää lähettämässä 1 Mb/s kohde palvelimeen tarkoittaa se sitä, että palvelimeen tulee 10 Gb:n kuorma joka sekunti. Tässä tapauksessa pätevät samat säännöt kuin normaalissa kuormassa. Tämän kaltaisessa tapauksessa halutaan enemmän palvelimia, jotka ovat jaoteltu useisiin datakeskuksiin. Näille palvelimille pitäisi olla kuormajako, joka jakaa kuorman tasaisesti jokaiselle palvelimelle. Kun kuorma jaetaan tasaisesti palvelimien välillä helpottaa se kuorman käsittelyä. (Lambert 2012.)

Kuitenkin nykypäivänä hyökkäysten koko voi kasvaa erittäin suuriin kokoluokkiin, jolloin ne saattavat olla paljon suurempia kuin yrityksen budjetti antaa myöden (Lambert 2012).

5.4.2 Ohjelmistosuunnittelu

Useimmat modernit web-sivut käyttävät paljon dynaamisia resursseja. Vaikkakin hyökkäyksen viemä kaista olisi hallittavissa, usein sivuston tietokanta tai räätälöidyt skriptit kaatuvat. (Lambert 2012.)

Tätä voidaan estää välimuistiin tallentavilla palvelimilla, joilla tarjotaan niin paljon staattista sisältöä kuin mahdollista. Web-välimuisti tarjoaa väliaikaisen varaston tiedon tallentamiseen, jolla vähennetään palvelimen kuormitusta ja kaistan käyttöä. (Lambert 2012.)

Kuten aikaisemmin DDoS-hyökkäyksistä mainittiin (osiossa, kuinka palvelunestohyökkäykset ovat muuttuneet), voi hyökkäys olla vain hämäystä muiden käytettävien reikien etsimiseksi. Tästä syystä olisi hyvä varautua DDoS-hyökkäyksien ohessa ilmestyviin tietomurtoihin jo ohjelmistosuunnittelun tasolla.

Tietomurtojen tai väärien käytösten estämiseksi jokainen ohjelmiston ulkopuolelta tuleva syöte olisi hyvä validoida. Käyttäjän syötteeksi pitäisi kelvata vain ohjelmaan määritetyt kelvolliset syötteet, ja kaikki muut syötteet tulisi olla epäkelpoja.

Lisäksi palvelun tulisi olla kerrosarkkitehtuurin eli MVC-mallin mukainen, jolloin sivulta ei ole suoraa yhteyttä tietokantaan, vaan kaikki tieto liikkuu kontrolleritason businesslogiikan mukaan.

Palvelussa tulisi myös välttää raskaita tietokantakyselyitä tai ainakin lukita ne kirjautumisen taakse. Raskaita tietokantakyselyitä ketjuttamalla voidaan palvelun takana toimiva tietokanta kaataa suhteellisen helposti.

5.4.3 Hyökkäyksen havainnointi- ja estojärjestelmät (IPS & IDS & WAF)

IPS & IDS

IPS normaalisti tarkkailee verkon liikennettä pakettien virratessa sen läpi. Se käyttäytyy samankaltaisesti kuin IDS yrittäen verrata pakettien tietoa IDP-tietokantaan, joka sisältää hyökkäyksissä käytettäviä pakettimuotojen allekirjoituksia. Se tutkii paketteja

ja etsii normaalista poikkeavaa toimintaa. IDS-toiminnan lisäksi IPS voi tehdä enemmän kuin vaan kirjata ja hälyttää. Se voidaan ohjelmoida reagoimaan siihen, mitä se tunnistaa. Reagointiominaisuus tekee siitä haluttavamman kuin IDS-ohjelmistosta. (McMillan 2009.)

IPS:n käytössä on kuitenkin vielä joitakin haittoja. IPS:t on suunniteltu estämään tietyn tyylistä liikennettä, jonka se voi tunnistaa mahdollisesti huonoksi liikenteeksi. IPS:llä ei ole ominaisuutta tunnistaa web-sovelluksen protokollalogiikkaa. Siksi IPS ei voi täysin tunnistaa, onko pyyntö normaali vai muunnettu sovellustasolla. Tämän takia se voi päästää jotkin hyökkäykset läpi tunnistamatta tai estämättä, erityisesti uuden tyyliset hyökkäykset, joista ei ole vielä IDP-tietokannassa tietoa eli allekirjoituksia. (McMillan 2009.)

Web-sovelluksia on suuri määrä, niin kaupallisia kuin kotitekoisiakin, sen takia hyökkäyksissä käytettäviä haavoittuvuuksia löytyy koko ajan lisää. IPS ei voi tehokkaasti kattaa kaikkia potentiaalisia haavoittuvuuksia ja itse asiassa saattaa jopa tuottaa enemmän virheellisiä esiintymiä. Virheelliset pakettihavainnot ovat todella huono asia, koska ne kuormittavat jo valmiiksi kiireistä turvallisuusanalysaattoria. Virheelliset osumat saattavat viivästää reagointia hyökkäyksiin tai päästää jopa hyökkäyksen täysin läpi, koska analysaattori yrittää vähentää kohinaa. (McMillan 2009.)

WAF

Yksi vaihtoehto hyökkäyksien estämiseen ja havainnointiin on WAF. WAF on suunniteltu suojaamaan web-sovelluksia ja palvelimia web-pohjaisilta hyökkäyksiltä, joita IPS ei voi estää. Samaan tyyliin kuin IPS, WAF voi olla verkko- tai isäntäpohjainen. Ne tutkivat sovelluksien tai palvelimien lähtevää ja tulevaa liikennettä. WAF:n ero IPS:ään tulee sen kyvystä tarkkailla 7. kerroksen web-sovelluksen logiikkaa eli sovellustasoa. (McMillan 2009.)

Kuviossa 5 nähdään OSI-mallin kerrokset johon McMillan (2009) artikkelissaan viittaa.

Siinä missä IPS vertaa liikennettä allekirjoituksiin ja poikkeavuuksiin, WAF tutkii käyttäjien syötteitä ja logiikkaa siitä, mitä on pyydetty ja mitä on palautettu. WAF suojaa

web-sovelluksia esimerkiksi SQL-injektiolta, cross-site scriptingiltä, session kaappauksilta, parametrien tai URL:en peukaloimiselta tai bufferin ylikuormittamiselta. Se tekee sen samaan tyyliin kuin IPS tutkimalla jokaisen paketin, joka tulee sisään tai lähtee ulos. (McMillan 2009.)

WAF toimitetaan normaalisti proxy tyylisesti web-sovelluksen eteen, joten se ei näe kaikkea liikennettä tietoverkossa. Liikenteen tutkimisella ennen kuin se saavuttaa sovelluksen WAF voi analysoida pyynnöt ennen kuin välittää ne eteenpäin. Tämä antaa sille suuren edun verrattuna IPS:iin. IPS:n tarkoitus on tutkia koko verkon liikennettä, sen takia se ei pysty tutkimaan sovellustasoa perusteellisesti. (McMillan 2009.)

WAF ei tunnista ainoastaan tunnettuja hyökkäyksiä, vaan se voi myös havaita ja estää uuden tyyppisiä hyökkäyksiä, joita ei ole aikaisemmin maailmalla esiintynyt. Tarkkailemalla epätavallista tai odottamattomia tapahtumia liikenteessä se voi hälyttää tai puolustaa tuntemattomia hyökkäyksiä vastaan. Esimerkiksi jos WAF havaitsee, että sovellus palauttaa paljon enemmän dataa kuin on odotettavissa, voi WAF blokata sen ja hälyttää jollekin. (McMillan 2009.)

Johtopäätökset

Web-sovellusten palomuurit ovat erityinen tapa, joita käytetään hyökkäysten havainnointiin. WAF on hyvä lisä IPS-järjestelmään tarjoten lisää suojaa hyökkäyksien torjumiseen. WAF:a voidaan käyttää ympäristöissä tuomaan vahvistettua turvaa web-sovelluksille tai palvelimille.

6 Käytetyt tekniikat

6.1 Python

6.1.1 Yleistä

Python on tulkattu, oliopohjainen, korkean tason ohjelmointikieli. Sen korkean tason datarakenteet yhdistettynä dynaamiseen kirjoittamiseen ja kiinnittämiseen tekevät siitä erittäin houkuttelevan vaihtoehdon nopeaan ohjelmistokehitykseen. Lisäksi Python on loistava väline skriptien kirjoittamiseen tai olemassa olevien komponenttien yhdistämiseen. Pythonin yksinkertainen ja helposti opittavissa oleva syntaksi on helppo lukea ja vähentää tällöin ylläpitokustannuksia. Python tukee useita moduuleita ja paketteja, jotka kannustavat modulaarisiin ohjelmistoihin ja koodin uudelleen käyttöön. Pythonin komentotulkki ja mittava kirjasto ovat saatavilla ilman maksua kaikille yleisille alustoille. (What is Python n.d.)

6.1.2 Miksi juuri Python?

Pythonia usein verrataan muihin tulkattuihin kieliin kuten Java, JavaScript, Perl, Tcl, tai Smalltalk.

Javaan verrattuna Python-ohjelmat usein pyörivät hitaammin, mutta ovat paljon nopeampia ohjelmoida. Python-ohjelmistot ovat usein 3-5 kertaa lyhempiä kuin vastaavat Java-ohjelmistot. Tämä erotus saadaan Pythonin korkeantason datarakenteista ja dynaamisesta kirjoitusrakenteesta. Esimerkiksi Python-ohjelmoijan ei tarvitse kuluttaa aikaa data-attribuuttien tai muuttujien tyyppien julistamiseen. (Comparing Python to Other Languages n.d.)

JavaScriptin tavalla Python tukee ohjelmointi tapaa, jolla voidaan käyttää yksinkertaisia funktioita ja muuttujia ilman luokkarakenteita. Toisaalta Pythonilla voidaan tehdä paljon suurempia kokonaisuuksia kuin JavaScriptillä, koska se tukee oliopohjaista ohjelmointitapaa. Luotaessa suuria ohjelmistokokonaisuuksia oliopohjainen luokkarakenne ja perintä ovat tärkeässä roolissa. (Comparing Python to Other Languages n.d.)

Kirjoitin kevään aikana valinnaiselle Internet of Things kurssille Pythonilla kasvojentunnistus ohjelmiston. Ihastuin Pythonin helppoon kirjoitustapaan, jossa koodin suoritusjärjestys merkitään sisennyksin. Tämä tekee pakostakin Python-ohjelmistosta helppolukuisen, koska sisennykset ovat aina kunnossa. Aluksi tämä aiheutti enemmän itkua kuin iloa, koska ohjelma ei toimi, jos sisennykset eivät ole kunnossa. Kuitenkin kun tekstieditorista tai kehitystyökalusta pakotettiin sisennyksissä käytettäväksi tabulaattorin sijasta välilyöntejä, ongelmat hävisivät.

Python valikoitui käytettäväksi kieleksi luonnollisesti. Halusin oppia kirjoittamaan Pythonilla enemmän, koska sitä käytetään kasvavissa määrin myös web-ohjelmonnissa.

6.2 Representional State Transfer – REST

6.2.1 Yleistä

REST tulee sanoista Representional State Transfer. REST on web standardeihin pohjautuva arkkitehtuuri, joka käyttää HTTP protokollaa tiedon välitykseen. Se tukeutuu ajatukseen, jossa jokainen komponentti on resurssi ja resurssiin pääsee yleisen HTTP rajapinnan kautta. REST esiteltiin vuonna 2000 Roy Fieldingin toimesta. (RESTful Web Services – Introduction n.d.)

REST-arkkitehtuurissa REST-palvelin tarjoaa yksinkertaisesti pääsyn resursseihin joihin REST-asiakas pääsee käsiksi. Jokainen resurssi on yksilöity URI:lla tai globaalilla tunnuksella. REST käyttää erilaisia esitystapoja esittääkseen tietoja. Näitä ovat esimerkiksi normaali teksti, JSON tai XML. Nykypäivänä JSON on kaikista suosituin formaatti web-palveluiden tiedon esittämiseen. (RESTful Web Services – Introduction n.d.)

6.2.2 Toiminta

Seuraavat yleisesti tiedetyt HTTP metodit ovat yleisesti käytettyjä REST pohjaisessa arkkitehtuurissa:

GET - Tarjoaa ainoastaan lukuoikeuden resurssiin.

POST – Käytetään uuden resurssin luomiseen.

DELETE – Käytetään resurssin poistamiseen.

PUT – Käytetään olemassa olevan resurssin muokkaamiseen tai korvaamiseen.

OPTIONS – Käytetään resurssin tuettujen komentojen hakemiseen.

Web-palvelu on kokoelma avoimia protokollia ja standardeja, joita käytetään tietojen vaihtamiseen sovellusten ja järjestelmien välillä. Ohjelmistosovellukset, jotka pyörivät useilla eri alustoilla tai ovat kirjoitettu useilla eri ohjelmointikielillä, voivat käyttää web-palveluita vaihtaakseen tietoa verkkojen välillä. (RESTful Web Services – Introduction n.d.)

REST-arkkitehtuuripohjaiset web-palvelut ovat tunnettuja nimellä ”RESTful web services”. Nämä web-palvelut käyttävät HTTP-metodeita REST-arkkitehtuurin implementoimiseen. REST-pohjainen web-palvelu yleensä määritää URI:n, joka tarjoaa tiedon esityksen esimerkiksi JSON muodossa ja tarjoaa lajitelman HTTP-metodeita. (RESTful Web Services – Introduction n.d.)

REST mahdollistaa ohjelmointikieliriippumattoman sovelluskehityksen, kunhan valittu kieli tukee REST-rajapintaa. Tämä mahdollistaa dynaamisten ohjelmistokokonaisuuksien kirjoittamisen vapaalla kielellä, joka hakee tietoja ja ohjaa haluttua resurssia HTTP metodien välityksellä.

Tässä opinnäytetyössä haettiin verkon analytiikkadataa JSON muodossa sFlow-RT:n REST-rajapinnasta ja haetun datan perusteella ohjattiin verkon liikennettä Floodlight-kontrollerin REST-rajapinnan kautta käyttäen HTTP GET, POST ja DELETE -metodeita.

6.3 Floodlight-kontrolleri

Floodlight on yritystason, Apache lisensoitu, Java perustainen avoimen lähdekoodin SDN-kontrolleri. Sitä tukee kehittäjien yhteisö, jonka joukossa on insinöörejä Big Switch Networkilta. (Floodlight Is an Open SDN Controller n.d.)

Miksi käyttää Floodlightia:

- **OpenFlow** – Floodlight toimii fyysisillä ja virtuaalisilla kytkimillä, jotka käyttävät OpenFlow-protokollaa.

- **Apache-lisensoitu** – Lisensointi antaa käyttää Floodlightia melkeinpä mihin tahansa tarkoitukseen.

- **Avoin yhteisö** – Floodlight on kehitetty avoimen yhteisön toimesta. Floodlight toivotaa tervetulleeksi koodillisen työpanoksen kaikilta aktiivisilta osallistujilta ja jakaa avoimesti tietoa projektin tilasta, seuraavista ominaisuuksista ja bugeista.

- **Helppo käyttää** – Floodlight on helppo pystyttää ja käyttää.

- **Testattu ja tuettu** – Floodlightia testataan ja parannetaan aktiivisesti.

(Floodlight Is an Open SDN Controller n.d.)

Juuri niin kuin Wilkins (2014) artikkelissaan mainitsee, Floodlight on järkevästi kirjoitettu ja skaalautuva SDN-kontrolleri. Lisäksi suurinta osaa sen ominaisuuksista voidaan ohjata REST API:a käyttäen. Tämä mahdollistaa sen, että kun SDN-verkolle tehdään ohjelmallista ohjausta, toteutuskieli on vapaa, kunhan se tukee REST-rajapintaa. Tällaisia kieliä ovat esimerkiksi, PHP, Python, Java, Javascript, Node.js, javaEE jne.

Floodlight-kontrolleri tuntui luontevalta sen helppokäyttöisyyden ja hyvän dokumentaation ansiosta. Alun perin toteutus oli tarkoitus tehdä Ryu-kontrolleria käyttäen, mutta sen kanssa törmättiin vain ongelmiin.

6.4 sFlow-RT

sFlow-RT sisältää InMonin tahdistamattoman analytiikkateknologian tarjoten reaaliaikaisen näkyvyyden SDN-verkkoon. Se mahdollistaa uusien suorituskyky tietoisten ohjelmistojen kuten kuormanjakamisen ja DDoS-suojauksen kehityksen. (sFlow-RT n.d.)

sFlow-RT-analytiikkakone saa jatkuvaa tietovirtaa sFlow Agenteilla, jotka ovat sulautettuna tietoverkon laitteisiin, isäntiin tai sovelluksiin. Se muuntaa tietovirran mittareiksi, joihin pääsee käsiksi RESTflow API:a käyttäen. RESTflow API tekee mittareiden hakemisen, raja-arvojen asettamisen, huomautusten saamisen helpoksi. Sovellukset voivat olla ulkoisia, kirjoitettuna millä tahansa kielellä, joka tukee HTTP / REST -kutsuja tai sisäisiä käyttäen sFlow-RT:een sulautettuja JavaScript/ECMAScript-kieliä. (sFlow-RT n.d.)

Tietoverkon näkyvyys on avainasemassa ymmärtämään vuorovaikutusta ulos skaalattujen palveluiden kesken pilvipalveluarkkitehtuurissa. Verkon, isännän ja sovellusten monitoroinnin yhtenäistäminen tarjoaa näkyvyyttä sovelluksiin, palvelimiin ja tietoverkon resursseihin, mitä tarvitaan ylläpitämään suorituskykyä. (sFlow-RT n.d.)

sFlow-RT-ohjelmisto oli työkalu, jolla saatiin selville tietoverkossa liikkuvien pakettien määrä. Pakettien määrästä johdettiin kuormanjako verkon eri osiin käyttäen Floodlight-kontrollerin REST-rajapintaa.

sFlow-RT tarjoaa verkon kuormasta reaaliaikaista tietoa. Tämä tieto on tarkasteltavissa joko html tai JSON muodossa. Verkon resurssia tarkasteltaessa html-muodossa sFlow-RT piirtää käyrää pakettien määrästä halutusta paikasta, johon on asennettu sFlow-Agentti. Sama tieto saadaan myös JSON-muodossa, jolloin se voidaan hakea Python ohjelmistoon HTTP GET -metodia käyttäen. Tämä mahdollistaa raja-arvojen asettamisen ohjelmistotasolla. Esimerkiksi jos pakettimäärä ylittää 500 pakettia sekunnissa, muutetaan verkon toimintaa jotenkin.

7 Toteutus

Opinnäytetyön toteutuksessa suunniteltiin ja toteutettiin kuormanjako-ohjelma, joka reagoi verkkoon tulevaan pakettiliikenteeseen ja jakaa sen tasaisesti verkon eri osiin. Toteutus tehtiin VMwaren SDN-Testbed-ympäristössä. Välitystaso eli verkkotopologia virtualisoitiin Mininet-ohjelmistolla. Kontrollitasossa käytettiin Floodlight-kontrolleria ja sovellustason toteutus tehtiin Python-ohjelmointikielellä.

7.1 Suunnittelu

7.1.1 Teoria

Opinnäytetyön päätavoitteena oli hallita verkon toimintaa ohjelmallisin menetelmin. Tästä johdettiin kuormanjako ohjelmisto.

Verkossa liikkuvan pakettiliikenteen havainnoimiseksi tarvittiin joku ohjelmisto, joka tarjoaa analytiikkadataa verkossa liikkuvista paketeista. Tähän on muutamia eri vaihtoehtoja: sFlow-RT, restFlow, sFlow-Trend jne.

Floodlight-kontrolleri tarjoaa REST-rajapinnan, mistä voidaan puskea uusia Flow-sääntöjä reitittimille. Nämä säännöt ohjaavat paketti liikennettä jostain lähteestä haluttuun osoitteeseen. Floodlight-kontrolleri käyttää oletuksena lyhintä mahdollista reittiä isäntien välillä. Tämä tarkoittaa sitä, että jos verkko on monimutkainen, jotkin reitittimet saattavat jäädä kokonaan käyttämättä. Kuorman jakamisen onnistumiseksi tulee muuttaa tätä reittiä isäntien välillä. Se onnistuu käyttämällä HTTP metodeita Floodlight-kontrollerin tarjoaman REST-rajapinnan välityksellä.

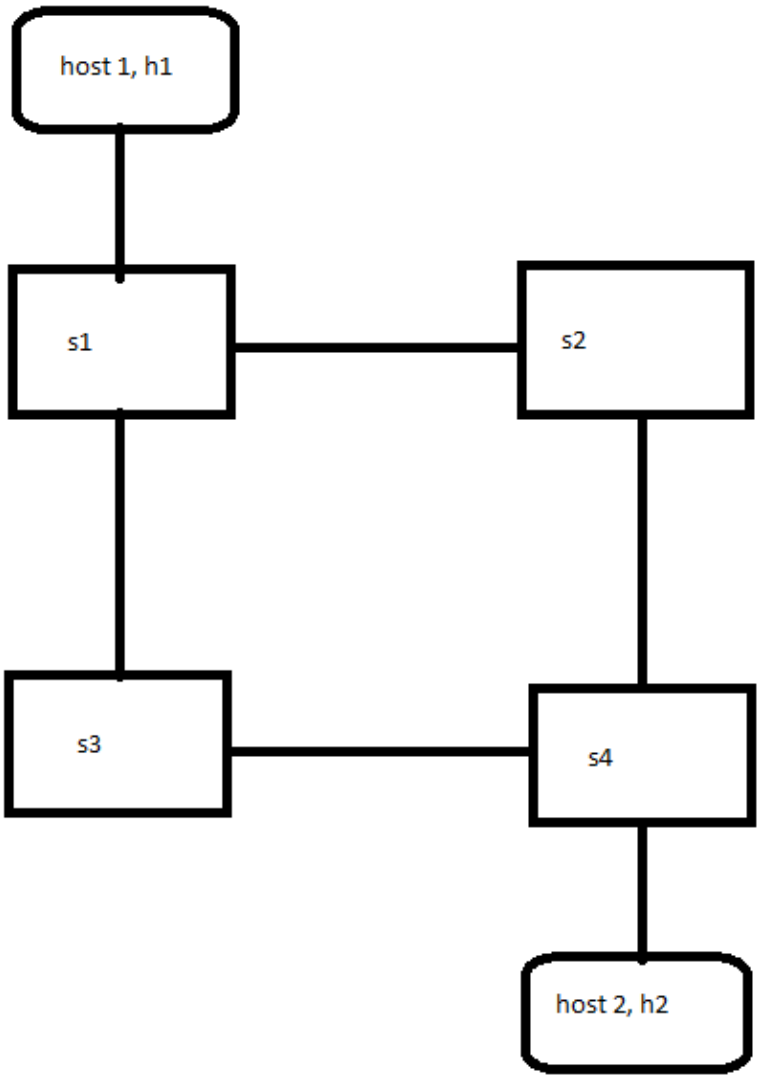
Vaikkakin Floodlight-kontrolleri itsessään on kirjoitettu Java-ohjelmointikielellä, sen tarjoama REST rajapinta mahdollistaa toisen ohjelmointikielen käyttämisen verkon ohjaamiseen. Sama pätee sFlow-RT ohjelmistoon, joka tarjoaa verkossa liikkuvan pakettidatan joko html- tai JSON-muodossa. JSON on yksi REST-rajapinnassa käytettävistä tiedostomuodoista, ja sen tiedot voidaan hakea HTTP GET -protokollaa käyttäen.

Kun verkossa liikkuva pakettien määrä haetaan JSON-muodossa sFlow-RT ohjelmistosta tasaisin väliajoin, voidaan siihen reagoida puskemalla verkkoon uusia sääntöjä

Floodlight-kontrollerin REST rajapintaa käyttäen HTTP POST -metodilla. Vastaavasti kun verkossa liikkuva pakettidata on tiedossa ja se laskee alle raja-arvon, voidaan säännöt poistaa verkosta HTTP DELETE -metodilla.

7.1.2 Kaaviot

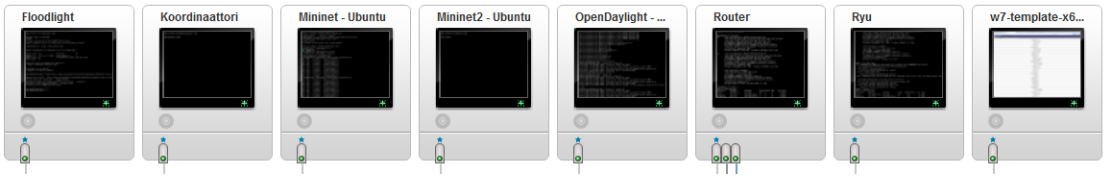
Kuvion 6 verkkotopologiassa nähdään opinnäytetyössä käytetyn välitystason topologia. Floodlight-kontrolleri käyttää automaattisesti laskettua lyhintä reittiä h1, s1, s2, s4, h2 pakettien välitykseen. Tällöin kytkin s3 jää kokonaan käyttämättä. Tavoitteena on tehdä ohjelmisto, jolla puolet s2:n läpi kulkevista paketeista ohjataankin s3 -reitittimen läpi. Tällöin todistetaan, että SDN-verkossa kuormaa on mahdollista jakaa verkon eri osiin.



Kuvio 6. Verkkotopologia

7.2 Ympäristön pystytys

Käytännön toteutus suoritettiin VMwaren SDN-testbed virtuaalisessa ympäristössä. Ympäristö luotiin valmiilta pohjalta, jolla SDN-verkkoa pystyttiin testaamaan. Ympäristöön kuului virtuaalikoneita, joihin työssä käytettäviä ohjelmistoja oli asennettu jo ennakkoon. Työssä käytettiin Floodlight-kontrolleria, sFlow-RT-ohjelmistoa, Mininet – Ubuntu-palvelinta sekä Windows 7 -tietokonetta. Työssä käytettävien koneiden IP-osoitteet ovat 192.168.2.104 (Floodlight), 192.168.2.101 (Mininet), 192.168.2.107 (Windows 7). Näiden lisäksi sFlow-RT-ohjelmisto asennettiin OpenDaylight-palvelimelle, jonka IP-osoite on 192.168.2.102. Kuviossa 7 nähdään koko ympäristö.



Kuvio 7. VMware SDN-testbed -ympäristö

Ensimmäisenä asennettin Floodlight-kontrolleri Floodlight-palvelimelle. Kuviossa 8 nähdään kontrollerin asennus.

```
floodlight@Floodlight:~$ cd floodlight
floodlight@Floodlight:~/floodlight$ java -jar target/floodlight.jar_
```

Kuvio 8. Floodlight-kontrollerin käynnistys

Kun kontrolleri jää kuuntelemaan verkkoa pitää käynnistää verkkotopologia Mininet – Ubuntu-palvelinta käyttäen. Mininet ei kuitenkaan oletuksena pitänyt sisällään opinnäytetyössä tarvittavaa verkkotopologiaa, joten se piti luoda itse. Kuviossa 9 nähdään verkkotopologian luomiseen käytettävä skripti, jonka kirjoittamisesta oli selkeä ohjeistus Mininetin dokumenttikirjastossa.

Skriptissä luodaan kaksi isäntää, H1 ja H2, ja neljä kytkintä S1, S2, S3, S4. Kun kytkimet ja isännät on luotu, lisätään reitit isäntien ja kytkimien väliin, jolloin saadaan kuviossa 6 piirretty verkkotopologia.

```
11  from mininet.topo import Topo
12
13  class MyTopo( Topo ):
14      "Simple topology example."
15
16      def __init__( self ):
17          "Create custom topo."
18
19          # Initialize topology
20          Topo.__init__( self )
21
22          # Add hosts and switches
23          H1 = self.addHost( 'h1' )
24          H2 = self.addHost( 'h2' )
25          S1 = self.addSwitch( 's1' )
26          S2 = self.addSwitch( 's2' )
27          S3 = self.addSwitch( 's3' )
28          S4 = self.addSwitch( 's4' )
29
30          # Add links
31          self.addLink( H1, S1 )
32          self.addLink( S1, S2 )
33          self.addLink( S1, S3 )
34          self.addLink( S2, S4 )
35          self.addLink( S3, S4 )
36          self.addLink( H2, S4 )
37
38
39  topos = { 'mytopo': ( lambda: MyTopo() ) }
```

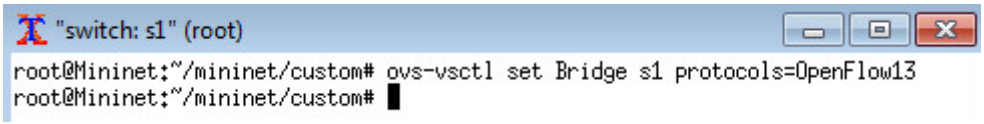
Kuvio 9. topo_custom.py

Kun skripti on valmis, käynnistetään verkko Windows 7 -koneelta SSH-yhteydellä, kuviossa 10 näkyvällä tavalla. Komennossa viitataan topo_custom.py-skriptiin ja kerrotaan, että topologia on mytopo. Samalla asetetaan verkolle kontrolleri, joka toimii Floodlight-palvelimella osoitteessa 192.168.2.104. Floodlightin oletusportti on 6653. Jos käynnistyskomentoon asettaa parametrin –x, käynnistää se automaattisesti xterm-istunnot isännille, kytkimille ja kontrollerille.

```
mininet@Mininet:~/mininet/custom$ sudo mn --switch=ovsk --custom topo_custom.py --controller=remote,ip=192.168.
2.104,port=6653 -x --topo=mytopo --mac
[sudo] password for mininet:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s4) (s1, s2) (s1, s3) (s2, s4) (s3, s4)
*** Configuring hosts
h1 h2
*** Running terms on localhost:10.0
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

Kuvio 10. Mininet-käynnistys

Kun mininet on käynnistetty, aukeaa sen CLI. Sillä voidaan antaa verkolle komentoja. Käynnistämisen jälkeen tulee asettaa jokaisen kytkimen OpenFlow-protokollaksi uusin versio eli OpenFlow1.3. Se asetetaan kuvion 11 mukaisella tavalla xterm-istunnossa.



Kuvio 11. OpenFlow-protokollan asetus

7.3 sFlow-RT ohjelman asennus

sFlow-RT-ohjelmisto asennettiin OpenDaylight-palvelimelle. Sen asentaminen ja käynnistäminen tapahtui kuvion 12 mukaisella tavalla.

```
wget http://www.inmon.com/products/sFlow-RT/sflow-rt.tar.gz
tar -xvzf sflow-rt.tar.gz
cd sflow-rt/
./start.sh
```

Kuvio 12. sFlow-RT-ohjelmiston asennus ja käynnistys

Kun sFlow-RT-ohjelmisto on asennettu, tulee seuraavaksi asentaa sFlow-agentit. Agentit keräävät verkosta tietoa, jolloin saadaan selville verkossa liikkuvat pakettimäärät. Ensimmäinen agentti asetettiin kytkimen s1-eth1 liitäntään kuviossa 13 näkyvällä tavalla. Tämän agentin avulla tiedetään, onko verkossa liikennettä ja kuinka paljon sitä on.

```
root@Mininet:~/mininet/custom# ovs-vsctl -- --id=@sflow create sflow agent=s1-e
th1 target=\"192.168.2.102:6343\" sampling=10 polling=20 -- set bridge s1 sflow
=@sflow
b03c7883-8d8e-4778-9193-749ee4db89db
root@Mininet:~/mininet/custom#
```

Kuvio 13. s1-eth1 Agentti

Toinen agentti asetettiin kytkimen s2-eth1 liitäntään kuviossa 14 näkyvällä tavalla. Tämän agentin avulla voidaan tarkkailla kytkimen s2 läpi kulkevaa liikennettä.

```
root@Mininet:~/mininet/custom# ovs-vsctl -- --id=@sflow create sflow agent=s2-e
th1 target=\"192.168.2.102:6343\" sampling=10 polling=20 -- set bridge s2 sflow
=@sflow
ee87f7c9-e218-4a25-9f5b-f52d0f20a90a
root@Mininet:~/mininet/custom#
```

Kuvio 14. s2-eth1 Agentti

Kolmas agentti asetettiin kytkimen s3-eth1 liitäntään kuviossa 15 näkyvällä tavalla. Vastaavasti taas tämän agentin avulla nähdään kytkimen s3 läpi kulkeva liikenne.

```
root@Mininet:~# ovs-vsctl -- --id=@sflow create sflow agent=s3-eth1 target=\"19
2.168.2.104:6343\" sampling=10 polling=20 -- -- set bridge s3 sflow=@sflow
30ee84b9-50c2-48e9-87bb-85e53736a253
root@Mininet:~#
```

Kuvio 15. s3-eth1 Agentti

Kun kaikki agentit on asetettu, nähdään verkossa liikkuvat paketit sFlow-RT-ohjelmiston web-käyttöliittymän avulla. Ne ovat saatavissa joko html-näkymänä, joissa ohjelmisto piirtää käyrää pakettiliikenteestä, tai JSON muodossa. Pakettien määrää käytetään kuormanjako ohjelmistossa määrittämään, koska kuormanjako aloitetaan.

8 Tulokset

8.1 Kuorman jakaminen SDN-verkossa

Opinnäytetyön toteutusosiossa tehtiin kuormanjako-ohjelmisto. Tässä luvussa näytetään ohjelman toiminta käytännössä. Piirretyt kuviot todistavat verkon jakamisen onnistumisen. sFlow-RT ei osaa piirtää kokonaiskäyrää kytkimen läpi kulkevista paketeista, joten se todistettiin kahta kuvaajaa ja ohjelmiston lokitiedostoa vertaamalla.

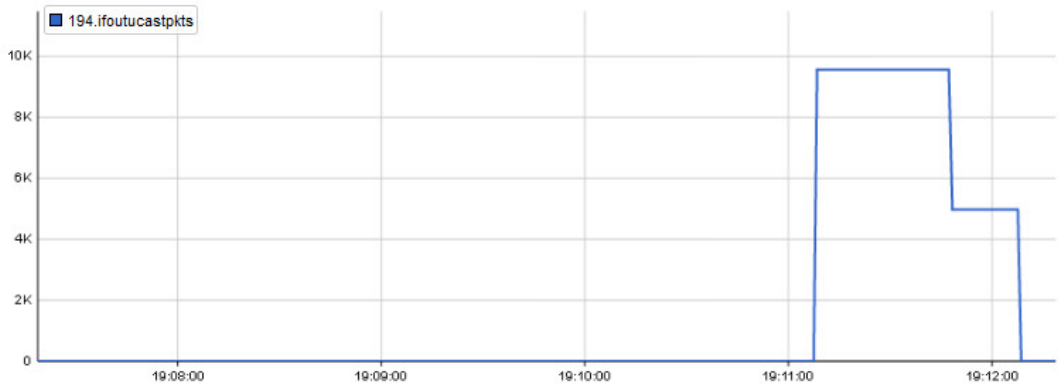
Testi aloitettiin lähettämällä ping-paketteja isännästä h1 isäntään h2. Parametri –f tekee tämän tulva metodilla, se lähettää paketteja isännästä h1, isäntään h2 niin nopeasti kuin suinkin on mahdollista. Tämä on yksi tapa tehdä DoS-hyökkäys. Metodi on näkyvillä kuviossa 16.

```
mininet> h1 ping -f h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
```

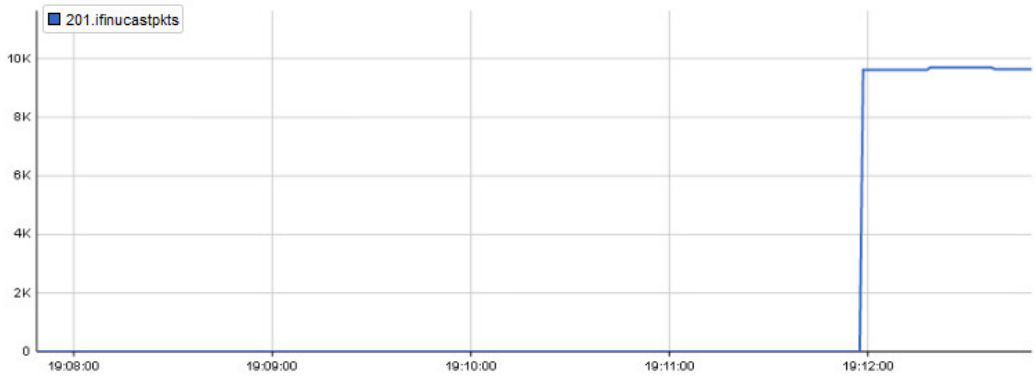
Kuvio 16. Ping flood h1, h2

Liitäntään s2-eht1 asetettu sFlow agentti näkee liitännän kautta sisään ja ulos menevät paketit. Kun ping-komento aloitetaan, kulkee paketteja molempiin suuntiin tämän liitännän kautta, koska Floodlight-kontrollerin oletusreitti on h1, s1, s2, s4, h2. Takaisintuloreitti on sama joskin toisessa järjestyksessä h2, s4, s2, s1, h1.

Kun kuormanjako ohjelmisto tekee muutoksen tietoverkon reititykseen, ulosmenevien pakettien tietovirta häviää kuviossa 17 näytetyllä tavalla. Tietoliikenne ilmestyy s3-eth1 liitäntään sisään tulevina paketteina kuvion 18 mukaisesti.



Kuvio 17. s2-eth1 ulos menevät paketit



Kuvio 18. s3-eth1 sisään tulevat paketit

Kuvioissa 18-19 huomataan kuinka, tietovirta vaihtaa paikkaa. Kuviosta 19 esitellyssä lokitiedostossa nähdään myös aikaleima, joka vastaa kuvion käyriin. Kello 19:11:07 puskettiin uudet reitityssäännöt verkkoon kuormanjako-ohjelmiston toimesta ja tehtiin siitä lokimerkintä. Tämä ilmestyi käyriin noin puolen minuutin viiveellä.

```
1  2016-05-02 17:18:06 Flow rules PUSHED and file created
2  2016-05-02 17:18:47 Flow rules DELETED
3  2016-05-02 17:38:44 Flow rules PUSHED
4  2016-05-02 17:40:26 Flow rules DELETED
5  2016-05-02 18:47:04 Flow rules PUSHED
6  2016-05-02 18:48:53 Flow rules DELETED
7  2016-05-02 18:50:05 Flow rules PUSHED
8  2016-05-02 18:51:53 Flow rules DELETED
9  2016-05-02 18:56:26 Flow rules PUSHED
10 2016-05-02 18:58:09 Flow rules DELETED
11 2016-05-02 19:00:07 Flow rules PUSHED
12 2016-05-02 19:02:39 Flow rules DELETED
13 2016-05-02 19:11:07 Flow rules PUSHED
14
```

Kuvio 19. BalanceLoader rulelog.txt

Kuviot 17, 18 ja 19 osoittavat, että kuorman jakaminen verkossa todellakin onnistui halutulla tavalla.

SDN-verkot ovat uusi ja kehittyvä ympäristö. Tässä työssä todistettiin, että kuorman jakaminen SDN-verkon eri osiin on mahdollista. Kun kuormanjako tulee voimaan, se vähentää kytkimelle s2 menevää kuormaa puolella. Vastaavasti kytkin s3, ei ole enää turhaan verkossa, vaan se jakaa kuorman kytkimen s2 kanssa.

Tästä päästään tulokseen, jossa verkon kuorma on jaettu tasaisesti kaikille verkkotopologiaan kuuluville laitteille.

8.2 Palvelunestohyökkäyksien tutkiminen

Palvelunestohyökkäykseen on siis mahdollista reagoida myös SDN-verkossa. Tässä toteutuksessa se todistettiin jakamalla kuormaa verkon eri osiin. Eli reagointi tapahtui kuormanjakometodilla. DDoS-hyökkäykseen tulisi kuitenkin reagoida verkossa eri tavalla, kuin tässä tehty toteutus. Silloin pakettiliikenteen ylittäessä sallitun raja-arvon tiputettaisiin ylimääräisiä paketteja lähettävän isännän liikenne kokonaan tai siihen ei enää vastattaisi.

Toinen mahdollisuus olisi jakaa liikennettä isäntien välillä. Tämä olisi kuitenkin vaatinut erilaisen verkkotopologian ja sitä olisi ollut vaikea havainnoida.

Kolmas mahdollisuus olisi ohjata pakettiliikenne NFV-palvelun kautta. Esimerkiksi kytkin s1 lähettäisikin paketin palvelimelle x, jossa olisi NFV-palvelukokonaisuus vaikkapa IPS ja WAF. Paketin ollessa kunnossa välitettäisiin se takaisin omaan verkkoon. Tämän kaltainen toteutus saattaa olla tulevaisuutta.

9 Pohdinta

Opinnäytetyön tavoitteena oli monitoroida SDN-verkon liikennettä ja ohjata sitä halutulla tavalla. Työssä käytetty Floodlight-kontrolleri osoittautui erittäin hyväksi vaihtoehdoksi. Sen älykäs logiikka ja hyvät rajapinnat tekivät sovellustason ohjelman tekemisestä helppoa.

Lisätavoitteena oli tutkia palvelunestohyökkäyksiä ja kuinka niitä voidaan estää. Opinnäytetyön teoriaosuudessa esiteltiin ratkaisuiksi ohjelmistosuunnittelu, IDS; IPS ja WAF. IPS, IDS ja WAF ovat kaikki NFV palveluita, jotka voidaan toimittaa haluttuihin paikkoihin verkossa.

NFV-palveluilla on tavoite luoda samanlaiset verkonhallintatyökalut SDN-verkkoihin, kun ne ovat rautapuolella. Samalla tämä uusi teknologia mahdollistaa ketterän verkkosuunnittelun ja tietoturvapalveluiden toimittamisen haluttuihin paikkoihin.

Mielestäni SDN-verkot ja NFV-palvelut ovat luomassa tietoverkkojen pariin ohjelmoinnista tuttua oliopohjaista ajattelua. Tulevaisuuden tietoverkoissa voidaan toimittaa esimerkiksi palomuureja dynaamisesti haluttuihin paikkoihin. Tätä voidaan verrata olion tai luokan uudelleen käyttämiseen ohjelmoinnissa.

Teoriassa SDN-verkossakin voidaan reagoida DDoS-hyökkäykseen. Tässä työssä esitelty kuorman jakaminen on yksi menetelmä. Kuitenkin tätä menetelmää tulisi käyttää hieman toisin. Kuorman jakaminen voidaan tehdä joko kytkimille tai isännille. Yleensä ne jaetaan niin sanottuihin clustereihin eli ryhmiin. Esimerkkinä voidaan käyttää vaikkapa kahta palvelinta: p1 ja p2. Palvelimet ovat yhtenä ryhmänä ja vastaavat samaan resurssiin. Kun palvelin p1 saa pyynnön, vastaa se siihen normaalisti; kun se saa seuraavan pyynnön, siihen vastaakin p2. Tällä tyylillä samaan ryhmään voidaan laittaa n kappaletta palvelimia, jotka vastaavat vuorotellen resurssiin tulevaan pyyntöön.

Kuorman jakaminen SDN-verkossa voidaan tehdä samalla lailla myös kytkimille. Tällöin tässä opinnäytetyössä käytetyt kytkimet s1 ja s4 jakaisivat paketit vuorotellen s2- ja s3-kytkimille. Tällöin lopputulos olisi täysin sama: kuorma jaettaisiin tasaisesti verkkoon.

WAF on loistava väline IPS:n tueksi. Kun IPS keskittyy OSI-mallin muihin osa-alueisiin, tutkii WAF sovellustason liikennettä. IPS:n toiminta perustuu hyökkäyksissä käytettyihin allekirjoituksiin, joita se etsii IDP-tietokannasta. Kuitenkin jos allekirjoitusta hyökkäyksestä ei ole olemassa, se saattaa päästä suoraan estojärjestelmien läpi.

Palvelunestohyökkäyksen päästessä läpi voitaisiin sitä opinnäytetyössä havaitulla tavoilla vähentää tai estää. Verkon liikennettä tarkkailtaisiin sFlow-monitorointidatan avulla. Liikenteen ylittäessä sallitun raja-arvon reagoisi verkko siihen pudottamalla normaalista poikkeavan pakettiliikenteen.

Näin ollen DDoS-migraatiota voidaan tehdä myös SDN-verkossa. DDoS-hyökkäystä ei kuitenkaan haluta päästää verkkoon asti, joten tätä tapaa ei suositella. Mielestäni SDN-verkon ohjelmoitavuus on myös mahdollisuus lisätä tietoverkon turvallisuutta. Esimerkiksi jos DDoS-hyökkäys sattuukin pääsemään estojärjestelmien läpi, siihen voidaan vielä verkkotasolla reagoida.

SDN-verkot tarjoavat uusia helpompia mahdollisuuksia verkon reitittämiseen ja suojaamiseen. Esittäisinkin SDN-verkon DDoS-migraatiota yhdeksi tasoksi tulevaisuuden verkkoturvallisuuteen.

Lähteet

Comparing Python to Other Languages. N.d. Artikkeli Pythonin verkkosivuilta. Viitattu 2.5.2016. https://www.python.org/doc/essays/comparisons/

Cyber Trust. N.d. Artikkeli Cyber Trust-hankkeesta Digilen sivustolla. Viitattu 12.5.2016. http://digile.fi/fi/palvelu/tutkimusohjelmat/cyber-trust/

Denial of service attack. N.d. Artikkeli Akamai Technologies Inc:n sivustolla. Viitattu 1.5.2016. https://www.akamai.com/uk/en/resources/denial-of-service-attack-dos.jsp

Denial of service attacks. N.d. Artikkeli Incapsula Inc:n sivustolla. Viitattu 29.4.2016. https://www.incapsula.com/ddos/ddos-attacks/denial-of-service.html

Floodlight Is an Open SDN Controller. N.d. Kuvaus Project Floodlight -sivustolla. Viitattu 12.5.2016. http://www.projectfloodlight.org/floodlight/

Garrison, S. 2014. Understanding the differences between Software Definen Networking, network virtualization and Network Functions Virtualization. Artikkeli Network Worldin verkkosivuilla. Julkaistu 11.2.2014. Viitattu 12.5.2016. http://www.network-world.com/article/2174268/tech-primers/understanding-the-differences-between-soft-ware-defined-networking-network-virtualizati.html

Lambert, P. 2012. DDoS attack methods and how to prevent or mitigate them. Artik-keli TechRebublic sivustolla. Julkaistu 15.10.2012. Viitattu 30.4.2016. http://www.techrepublic.com/blog/it-security/ddos-attack-methods-and-how-to-pre-vent-or-mitigate-them/

McMillan, J. 2009. IDFAQ: What is the difference between an IPS and a Web Application Firewall?. Arikkeli SANS Instituten sivustolla. Julkaistu 1.11.2009. Viitattu 30.5.2016. https://www.sans.org/security-resources/idfaq/what-is-the-difference-between-an-ips-and-a-web-application-firewall/1/25

Podrezov, A. 2004.Threat descripton DOS. F-Secure 24.5.2004. Viittattu 1.5.2016. https://www.f-secure.com/v-descs/dostool.shtml

RESTful Web Services – Introduction. N.d. Tutorialspoint verkkosivujen ohje. Viitattu 2.5.2016. http://www.tutorialspoint.com/restful/restful_introduction.htm

sFlow-RT. N.d. Tuotekuvaus inMon.com verkkosivuilla. Viitattu 12.5.2016. http://www.inmon.com/products/sFlow-RT.php

Software-Defined Networking: The New Norm for Networks. 2012. Open Networking Foundation 13.4.2012. Viitattu 29.4.2016. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

Traffic Monitoring using sFlow. N.d. sFlow.org sivuston pdf esitys. Viitattu 12.5.2016. http://www.sflow.org/sFlowOverview.pdf

What is Python? N.d. Executive summary. Artikkeli Pythonin verkkosivuilta. Viitattu 2.5.2016. https://www.python.org/doc/essays/blurb/

Wilkins, S. 2014. A Guide To Software Defined Networking (SDN) Solutions. 2014. Artikkeli tom's IT PRO -sivustolla. Julkaistu 25.11.2014. Viitattu 26.4.2016. http://www.tomsitpro.com/articles/software-defined-networking-solutions,2-835-2.html

Wilson, J. 2015. Delivering Security Virtually Everywhere with SDN and NFV. IHS Infonetics komiteamietintö Juniper Networksin sivuilla. Julkaistu 1.7.2015. Viitattu 12.5.2016. https://www.juniper.net/assets/kr/kr/local/pdf/industry-reports/2000601-en.pdf

Liitteet

Liite 1. Kuormanjako-ohjelmisto

```
import httplib
import json
import urllib
import urllib2
import requests
import json
from pprint import pprint
import time
import datetime
import os.path

#This is load balancing script used to do custom routing in SDN-
network
#Author Asmo Korkiatupa

#maaritetään funktio loadBalance
def loadBalance():
    #asetetaan apumuuttuja
    pushed = False

    #haetaan verkossa liikkuvat paketit yhden sekuntin intervallilla
while loopissa
    while True:
        response = requests.get('http://192.168.2.102:8008/met-
ric/ALL/ifinucastpkts/json')
        data = response.json()

        #parsitaan vastauksesta metricValue
        for i in data:
            metric = i['metricValue']
        print metric

        #timestamp on aikaleima muuttuja jota käytetaan lokitiedon
kirjoituksessa
        timestamp = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
        time.sleep(1)

        #jos verkossa liikkuu yli 1000 pakettia / sekunti muutetaan
reititysta.
        if metric >= 1000:

            if pushed == False:
                #pusketaan flow saannot floodlight-kontrollerille
                pusher.set(flow1_1)
                pusher.set(flow1_2)
                pusher.set(flow1_3)
                pusher.set(flow2_1)
                pusher.set(flow2_2)
                pusher.set(flow3_1)
                pusher.set(flow3_2)
                pusher.set(flow4_1)
                pusher.set(flow4_2)
                pusher.set(flow4_3)

                print "Kuorman jako otettu kayttoon"
                pushed = True
```

```
                #jos lokitiedosto on jo olemassa lisataan sinne rivi
ja aikaleima
                if os.path.isfile("rulelog.txt"):
                    #avataan tiedosto lisataan seuraava rivi para-
metri "a" tarkoitta appendia.
                    file = open("rulelog.txt","a")
                    file.write(timestamp+" Flow rules PUSHED\n")
                    file.close()
                    #tiedosto suljetaan.
                #jos lokitiedostoa ei ole olemassa sellainen luodaan.
                else:
                    #lokitiedostoon kirjoitus ja rivin vaihto \n
                    file = open("rulelog.txt","w")
                    file.write(timestamp+" Flow rules PUSHED and file
created\n")
                    file.close()
                    #tiedoston sulkeminen
            #jos saannot floodlight-kontrollerille on jo puskettu ja
kuorma yli 1000pkt/s
            # tulostetaan jako kaytossa
            if pushed == True and metric >=1000:

                print "jako kaytossa"
        #jos floodlight-kontrollerille on puskettu flow saannot ja
value alle 1000
        if pushed == True and metric <=1000:

            # poistetaan flow saannot, kutsutaan staticflow pusherin
remove metodia
            pusher.remove('application/json',flow1_1)
            pusher.remove('application/json',flow1_2)
            pusher.remove('application/json',flow1_3)
            pusher.remove('application/json',flow2_1)
            pusher.remove('application/json',flow2_2)
            pusher.remove('application/json',flow3_1)
            pusher.remove('application/json',flow3_2)
            pusher.remove('application/json',flow4_1)
            pusher.remove('application/json',flow4_2)
            pusher.remove('application/json',flow4_3)
            #apumuuttuja falseksi
            pushed = False
            print "Kuormanjako poistettu"
            #kirjoitetaan lokitiedostoon, etta kuorman jakaminen
poistettu
            file = open("rulelog.txt","a")
            file.write(timestamp+" Flow rules DELETED\n")
            file.close()
            #tiedosto suljetaan.

#static flow pusher luokka
class StaticFlowPusher(object):
    #kaytetaan uuden instanssin luomisessa
    def __init__(self, server):
        self.server = server

    def get(self,data):
        ret = self.rest_call({}, 'GET')
        return json.loads(ret[2])
    #kaytetaan flow saantojen puskemiseen
    def set(self,data):
        ret = self.rest_call(data,'POST')
        return ret[0] == 200
    #flow saantojen poistaminen
```

```
    def remove(self, objtype, data):
        ret = self.rest_call(data, 'DELETE')
        return ret[0] == 200
    #rest kutsun muotoilu
    def rest_call(self,data,action):
        path = '/wm/staticflowpusher/json'
        headers = {
            'Content-type': 'application/json',
            'Accept': 'application/json',
            }
        body = json.dumps(data)
        conn = httplib.HTTPConnection(self.server, 8080)
        conn.request(action, path, body, headers)
        response = conn.getresponse()
        ret = (response.status, response.reason, response.read())
        print ret
        conn.close()
        return ret
#luodaan uusi ilmentyma staticflowpusherista
#parametriksi asetetaan floodlight-kontrollerin ip osoite.
pusher = StaticFlowPusher('192.168.2.104')

#verkkoon puskettavat flow saannot
#kytkin 01 eli s1
flow1_1 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_1_1",
    "cookie":"0",
    "priority":"32768",
    "in_port":"1",
    "set_ipv4_src":"10.0.0.1",
    "set_ipv4_dst":"10.0.0.2",
    "active":"true",
    "actions":"output=2"
    }
flow1_2 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_1_2",
    "cookie":"0",
    "priority":"32768",
    "in_port":"2",
    "set_ipv4_src":"10.0.0.2",
    "set_ipv4_dst":"10.0.0.1",
    "active":"true",
    "actions":"output=1"
    }
flow1_3 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"flow_mod_1_3",
    "cookie":"0",
    "priority":"32768",
    "in_port":"3",
    "set_ipv4_src":"10.0.0.2",
    "set_ipv4_dst":"10.0.0.1",
    "active":"true",
    "actions":"output=1"
    }
#kytkin 04 eli s4
flow4_1 = {
    'switch':"00:00:00:00:00:00:00:04",
    "name":"flow_mod_4_1",
    "cookie":"0",
    "priority":"32768",
```

```
    "in_port":"3",
    "set_ipv4_src":"10.0.0.2",
    "set_ipv4_dst":"10.0.0.1",
    "active":"true",
    "actions":"output=2"
    }
flow4_2 = {
    'switch':"00:00:00:00:00:00:00:04",
    "name":"flow_mod_4_2",
    "cookie":"0",
    "priority":"32768",
    "in_port":"2",
    "set_ipv4_src":"10.0.0.1",
    "set_ipv4_dst":"10.0.0.2",
    "active":"true",
    "actions":"output=3"
    }
flow4_3 = {
    'switch':"00:00:00:00:00:00:00:04",
    "name":"flow_mod_4_3",
    "cookie":"0",
    "priority":"32768",
    "in_port":"1",
    "set_ipv4_src":"10.0.0.1",
    "set_ipv4_dst":"10.0.0.2",
    "active":"true",
    "actions":"output=3"
    }
#kytkin 03 eli s3
flow3_1 = {
    'switch':"00:00:00:00:00:00:00:03",
    "name":"flow_mod_3_1",
    "cookie":"0",
    "priority":"32768",
    "in_port":"1",
    "active":"true",
    "actions":"output=2"
    }

flow3_2 = {
    'switch':"00:00:00:00:00:00:00:03",
    "name":"flow_mod_3_2",
    "cookie":"0",
    "priority":"32768",
    "in_port":"2",
    "active":"true",
    "actions":"output=1"
    }
#kytkin 02 eli s2
flow2_1 = {
    'switch':"00:00:00:00:00:00:00:02",
    "name":"flow_mod_2_1",
    "cookie":"0",
    "priority":"32768",
    "in_port":"1",
    "set_ipv4_src":"10.0.0.1",
    "set_ipv4_dst":"10.0.0.2",
    "active":"true",
    "actions":"output=2"
    }

flow2_2 = {
    'switch':"00:00:00:00:00:00:00:02",
```

```
    "name":"flow_mod_2_2",
    "cookie":"0",
    "priority":"32768",
    "in_port":"2",
    "set_ipv4_src":"10.0.0.2",
    "set_ipv4_dst":"10.0.0.1",
    "active":"true",
    "actions":"output=1"
    }
#suorittaa ohjelman ajamisen. eli kutsuu funktiota loadBalance
#loadBalance funktion sisalla looppi joten pyorii kunnes sammutetaan
ctrl+c
loadBalance()
```